This article originally appeared in the January 2006 edition of Better Software magazine.
http://www.stickyminds.com

Bulking Up

Fiona Charles

Testers are almost always part of a team. As the only tester on a project team, or as a member of a test team that is part of a larger project team, we need to interact with our colleagues to get our jobs done. But for some testers, that interaction—the people skills, or soft skills—is their weak spot.

A tester needs to strengthen soft skills just like an athlete needs physical conditioning. A successful basketball player must have certain “technical” skills: dribbling, passing, and shooting. But without a foundation of general physical fitness and agility, a player will not make the best use of those skills and will never be a star. For an athlete, physical fitness is a basic requirement and an enabler for technical s kills.

As a tester you are a knowledge worker, making your living by applying technical skills. Instead of the physical fitness skills needed by an athlete, you need soft skills to empower and put your technical skills to work. As you build up your people skills and enhance your ability to communicate, you are bulking up on useful skills that will shape you into a top-form tester.

Two specific people skills—teamwork and communication—are basic skills that every successful tester, even the most junior, needs. Unfortunately, these people skills are often neglected in testers’ education plans.

Teamwork Toning

While teamwork is critical for everyone in IT, testers have unique challenges as team members. A tester must be seen as a solid contributor to the collective effort but must also maintain a level of independence sufficient to complete the testing. Achieving this crucial balance is difficult, and soft skills are essential to help you do it.

At some point in their careers, most testers hear statements like these:

- “That’s not a fair test! Nobody would do something like that.”
- “You’re asking too many questions. The developers are complaining it’s taking too much of their time.”
- “You’re just not a team player.”

As a tester you are an anomaly. The project’s end product is a working software system, which most of the project team members are devoting all their efforts to building. Then you come along and expose the flaws.

Your job (perhaps your secret delight) is to break the software everyone else is working hard to create. If you test well, you will probably find many bugs that will take time to fix and retest. When you report those bugs, you are providing valuable information, but your work may make developers feel exposed and threatened. It is not a large leap for developers, particularly on an inexperienced team, to start seeing you as an obstacle rather than a helper. If the development lead and project manager feel the same way, your testing skills alone can't save you.

So your tester role puts you at risk of becoming a pariah or the last person left when the teams are picked. You end up in a precarious situation where someone is asking you to suppress information:

- "Look, you don't need to log these bugs. I'll just fix them right now."
- "All this stuff about so-called risks is just negative. That sort of talk is not helping the project."
- "We're all a team here. It doesn't make sense for you to report a red status on your part of the project when everyone else is green."

It can be tempting—appealing to your sense of camaraderie and teamwork. Or it can be coercive, with someone's refusal to hear you or trying to shame you into silence. If you succumb, you will compromise your effectiveness as a tester.

To combat these obstacles, you need people skills in order to exercise your testing skills. Ideally, you will be able to interact with others so that your position on the team is unquestioned. When you cannot avoid "teamwork issues," your credibility will depend on your soft skills.

Have a Game Plan

What can you do? The best strategy is to present yourself from the beginning as a dedicated team member with a specific job that adds value and contributes to the team's goals. It helps to remember a few simple tips.

- Have a clear understanding of the tester's role.
  - Your job is to provide information by surfacing facts and reporting them, not by suppressing information.
  - You are neither the quality gatekeeper nor the quality judge and jury. Never be judgmental about bugs or other problems, even if—or especially when—the quality is poor.
- Ask how you will be measured, and ask for regular feedback on your performance.
- Assess the culture of the team and find ways to fit in.
  - Identify the principal objective and make it your objective. Although this may sound obvious, it's hard for most testers to put another objective ahead of quality and really mean it. If meeting the schedule is paramount, you need to focus on finding bugs that might threaten the schedule.

    - Use the project objective as the frame of reference for your important conversations. If you talk in terms of absolute quality when the project driver is the schedule, your message will seem irrelevant. Instead, talk about project issues in relation to schedule risk.
    - Make allies rather than enemies.
        - Try to understand what motivates other team members. Are developers rewarded for "completing" development on time, regardless of whether they have tested their code?
        - Offer to help where you can, but never neglect your own job.
        - Don't drop public bombshells that could embarrass people or catch them by surprise.
        - Don't try to score points in meetings. If your testing is blocked by bugs, talk to the developer or development lead privately and enlist her help. Then you can report that you are working together to resolve the problem.

- Learn to negotiate.
    - Stand up for what you believe in, but pick your battles. It helps to have a mental framework like project risk for deciding which battles are crucial. If the schedule is the project driver, then evaluate each potential battle in relation to schedule risk. Is this issue important enough to delay the project?
    - Negotiate honestly. Propose a reasonable conclusion that is in the best interest of the project. People begin negotiations with unreasonable demands in case they need to negotiate down, but this can foster bad feelings and injure your credibility.
    - Avoid adversarial negotiation. Offer win-win propositions.

Communication Conditioning

Next to teamwork, good communication is the most important soft skill a tester can have.

Written Communication

Like it or not, writing is a requirement of your job as a tester. Often, a defect report is your first and most challenging writing task. Entry after entry comes back with "Not enough information" or "Can't reproduce," yet you know the problem is readily reproducible.

Because defect reports are central to every tester's job, this is an excellent place to improve your writing skills. Ask specific questions oriented around the three Cs:

- ❑ **Clarity**: Is this a clear, unambiguous strategy/report/plan, written in a logical order?
    - Write using simple terms.
    - Do not leave room for misinterpretation.
    - Start by summarizing the most important information, and then give supporting detail in logical order.
    -
- ❑ **Completeness**: Is there anything missing?
    - Describe what happened, and why it is a problem.
    - Include the information required to explain how you found the problem, including the data used and the steps followed.

    - Demonstrate the problem with screen shots or other evidence.

- **Concision**: Does it contain extraneous information or is the report verbose?
    - Focus on essentials.

Someone will read every project document you write. Improve your reports by asking developers for feedback and incorporating their suggestions. This will make the lives of the developers easier and encourage them to be helpful with your reports. Writing, like programming, requires debugging and testing.

Practice is essential. Write something every day, and use the three Cs to examine your work critically. Write about anything—from impressions of the day's testing to observations of the subway ride home. Set each piece aside for a few days and then review it. Does it say what you wanted to say? How could you make it more clear, complete, and concise?

Finally, read on any subject that interests you by authors whose writing you admire and try to understand why their writing works. What makes a particular book or article clear, interesting, and easy to read?

Oral Communication

You can flex your communication skills while speaking, too. When you speak up in a status meeting or talk to a project manager about what you have not been able to cover in your testing, you need to be clear, complete, and concise.

Many everyday occasions require a tester to have good oral communication skills:

- Asking questions about the software and interpreting the answers
- Interviewing for jobs or project roles
- Presenting testing status in a project meeting

Each of these requires you to communicate your meaning accurately. You also need to hear what others are saying and respond appropriately.

Stay In-Bounds

If you dive straight into the details or wander off the point, you will lose your audience. Start by summing up the critical information. That may lead to questions or a request for supporting details. Only when you are sure your audience has understood the main point—and is willing to hear more—should you move out into amplifying your message. It is essential to stick to the point and deliver the details in a logical sequence. A tester who can tell a compelling story in this way is more likely to be heard.

Practice, by writing down the most important points ahead of a meeting. Summarize each point concisely without obscuring the message and rank the points in order of importance. After the meeting, review how well you did. Did the participants quickly grasp the essential information? Were there questions you could have anticipated? Were people impatient with your explanations? What could you improve next time?

Ask yourself the same questions about impromptu conversations. If someone requests information and seems dissatisfied with your answers, you can also ask them how you can improve your delivery.

Endurance Training: Listening
Listening is equally important. Anytime you talk—and often even when you don't—you should be prepared to listen.
As in exploratory testing, the answer to one question will lead you to the next question. Even if you have brought a prepared set of questions to the interview, it's dangerous to stick to your agenda when a developer or architect is telling you other (perhaps important) things about the software. Ask clarifying questions, delve deeper on a particular thread, broaden the inquiry—all these depend on your ability to hear what is really being said.

Listening to questions is as essential as listening to answers. This should be obvious in a job interview, but some testers come to interviews and don't really hear the questions. If I'm faced with a candidate's failing to answer my questions after a couple of tries, I terminate the interview. I know I will have a frustrating time with a poor listener on my team, no matter how wonderful their testing skills are. A solid contender for the job will admit he or she doesn't understand a question or ask for clarification before answering thoughtfully.

The need to listen is perhaps less obvious in a status meeting, but a tester cannot afford to ignore everyone else's status. Your effectiveness depends on knowing what is going on elsewhere in the project. Listening to others, observing their body language, hearing what they are saying and what they are not saying, and asking pertinent questions—all give you crucial clues about the state of the software. A project meeting provides an opportunity to pick up other people's perceptions about the testing or testers and possibly influence them for the better.

Informal conversations with other project members are another common situation where listening skills are important for a tester. Typically, developers' chitchat is far in advance of official news and can be invaluable in directing your testing. Listen carefully and apply a reasonableness filter.

You can practice listening in various ways. Build in a feedback step to verify that you heard correctly. Offer to take meeting minutes and request feedback from participants before final distribution. Summarize the key points of conversations and play them back. During a conversation, you can do this before moving to the next point. Say something like, "I'd like to play that back to make sure I understand what you're saying."

Playing back important conversations in writing will ensure that you and the other participants agree about what was said and prevent any later misunderstanding.

Summarize the key points in an email and preface your message with something like, "The following summarizes my understanding of our conversation. Please let me know of any omissions, errors, etc."

Review the comments that come back. If you missed or misinterpreted important information, ask yourself how that happened. Were you too focused on thinking about your next point to really hear the other person? Were you trying to fit what you heard into a preconceived model? These are common causes of poor listening. Keep notes about what you learn, and review the notes regularly as a reminder of what to watch out for.

Sweat It Out: Difficult Conversations
Sooner or later, every tester will have to deal with more difficult conversations.

- Defending your test strategy to skeptical managers or developers

- Explaining or justifying why testing is taking so long

Challenging situations like these can happen on any size project. Project managers, architects, and developers sometimes hold firm opinions about the tester's place on a project and just how, or how much, you should test.

Occasions will arise when you have to deliver unwelcome news. One of the most difficult conversations you can have as a responsible tester occurs when you have to persuade managers that the software you have been testing is not ready, and it would be a better use of everyone's time to send it back to development.

Some tips for difficult conversations:

- Be prepared.
  - Have a clear sense of your vision of testing and the value it brings to the project.
  - Keep track of your work and be prepared with facts to demonstrate coverage and progress.
  - Prepare and file a weekly status report, even if it's not a project requirement. At the very least, clearly outline what you did, what you found, and any issues impeding progress. If nobody wants it, do it for yourself.

- Interact with the others in a conversation.
  - Listen carefully to the other person's point of view and ask for clarifications where needed.
  - Deal with questions directly. If you don't know the answer to a particular question, say that you will investigate and come back with the answer.
  - Acknowledge the importance of the other person's viewpoint. If you can't honestly do that, at least acknowledge the strength of his convictions or feelings, as in, "I understand that this is important to you."

- Always try to sound cooperative and positive.
  - Watch your tone. It can be difficult to avoid sounding impatient or abrupt when you see your point of view as a self-evident truth, but you will be more persuasive if you are open.
  - Watch the other person's reactions. If you sense resistance, you may be coming on too strong. Make sure you explain your points, and don't patronize others if they take a little time to get there.

- Try to maintain an even emotional keel, especially when a project becomes stressful. If it helps to think you wouldn't be employed without software bugs, then do that.

- Ask for time if you need it.
  - If you are caught by surprise, say something like, "This is an important conversation, and I want to do it justice. May I have a little time to prepare?"

Conclusion

Soft skills are not only skills in and of themselves but also a critical part of your mental conditioning as a master of technical skills. Developing teamwork and communication skills takes practice and patience. As you develop and nurture all of your skills—technical and non-technical, hard and soft—you will find that you make less and less distinction between them. Together, they are all skills that will make you a more effective tester.

Some Tips for Further Reading

Many excellent books and articles go into depth on all the topics covered here. Following is a brief selection of sources that are not only useful but also highly readable examples of good writing.

- Communication
  - Karten, Naomi. *Communication Gaps and How to Close Them.* New York: Dorset House Publishing, 2002.
  - Emery, Dale H. *Untangling Communication.* www.dhemery.com/articles/untangling_communication, originally published in STQE, Volume 3, Issue 4.
- Writing
  - Strunk, William Jr., and E.B. White. *The Elements of Style*. New York: Longman, 4th edition, 2000.
  - Weinberg, Gerald M. *Weinberg on Writing: How to Finish Your Project Using the Fieldstone Method.* New York: Dorset House Publishing, 2005.

- Negotiation
  - Fisher, Roger, and William Ury. *Getting To Yes: Negotiating Agreement Without Giving In*. New York: Penguin Group (USA), 1981. This very useful book focuses on negotiating while building positive relationships.

- Tracking testing
  - A useful example is James Bach's Testing Dashboard, available on his Web site www.satisfice.com.

- Other examples of good writing in our field
  - Anything by Gerald M. Weinberg, but especially *Quality Software Management*, *Volumes 1-4.* New York: Dorset House Publishing.
  - Books by Robert Glass, Tim Lister, and Tom De Marco.